Sato et al
Filed 7/22/03
Q 76641
1 of 1

日　本　国　特　許　庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２００２年　７月２３日

出願番号
Application Number: 特願２００２－２１３８９５

[ST.10/C]: [ＪＰ２００２－２１３８９５]

出願人
Applicant(s): ＮＥＣエレクトロニクス株式会社

２００３年　５月１３日

特許庁長官
Commissioner,
Japan Patent Office 太田信一郎

出証番号　出証特２００３－３０３４３３２

【書類名】　特許願
【整理番号】　74510257
【あて先】　特許庁長官殿
【国際特許分類】　G06F 17/50
【発明者】
【住所又は居所】　東京都港区芝五丁目７番１号　日本電気株式会社内
【氏名】　佐藤　光一
【発明者】
【住所又は居所】　東京都港区芝五丁目７番１号　日本電気株式会社内
【氏名】　澁谷　洋志
【発明者】
【住所又は居所】　東京都港区芝五丁目７番１号　日本電気株式会社内
【氏名】　黒坂　均
【特許出願人】
【識別番号】　000004237
【氏名又は名称】　日本電気株式会社
【代理人】
【識別番号】　100103894
【弁理士】
【氏名又は名称】　家入　健
【手数料の表示】
【予納台帳番号】　106760
【納付金額】　21,000円
【提出物件の目録】
【物件名】　明細書　1
【物件名】　図面　1
【物件名】　要約書　1
【包括委任状番号】　0118499
【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　システムレベル設計方法及びシステムレベル設計装置

【特許請求の範囲】

【請求項１】

コデザイン装置によってシステムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するステップと、

前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報をトップレベル記述生成装置に入力するステップと、

前記トップレベル記述生成装置において入力された情報に基づいてクロックベースシミュレーション用の記述を自動生成するステップとを備えたシステムレベル設計方法。

【請求項２】

前記クロックベースシミュレーション用の記述を自動生成するステップは、前記アドレス情報に基づき、バスとＣＰＵインターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成するステップを含むことを特徴とする請求項１記載のシステムレベル設計方法。

【請求項３】

前記クロックベースシミュレーション用の記述を自動生成するステップにおいては、仮想バスによりバス接続の記述が行われていることを特徴とする請求項１又は２記載のシステムレベル設計方法。

【請求項４】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するステップを備え、

このステップは、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、

検出された場合に当該グローバル変数をポートに置き換えるステップを有する

ことを特徴とする請求項１、２又は３記載のシステムレベル設計方法。

【請求項５】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計方法であって、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、

検出された場合に当該グローバル変数をポートに置き換えるステップを備えたシステムレベル設計方法。

【請求項６】

システムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するコデザイン装置と、

前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報を入力し、入力した情報に基づいてクロックベースシミュレーション用の記述を自動生成するトップレベル記述生成装置を備えたシステムレベル設計装置。

【請求項７】

前記トップレベル記述生成装置は、クロックベースシミュレーション用記述を自動生成する場合に、前記アドレス情報に基づき、バスとＣＰＵインターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成することを特徴とする請求項６記載のシステムレベル設計装置。

【請求項８】

前記トップレベル記述生成装置は、仮想バスによりバス接続の記述を行うことを特徴とする請求項６又は７記載のシステムレベル設計装置。

【請求項９】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換する手段をさらに備え、

この手段は、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れ

る構成を検出する手段と、

検出された場合に当該グローバル変数をポートに置き換える手段とを有することを特徴とする請求項６、７又は８記載のシステムレベル設計装置。

【請求項１０】

システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計装置であって、

アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出する手段と、

検出された場合に当該グローバル変数をポートに置き換える手段を備えたシステムレベル設計装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、大規模回路を設計するためのシステムレベル設計方法及びシステムレベル設計装置に関する。

【０００２】

【従来の技術】

近年、ＡＳＩＣ、ＣＰＵ、メモリ等から構成される複合演算処理システムの開発に対する要求が高まっている。このようなシステムを設計する場合、システム全体の処理動作を記述したアルゴリズムを人手により作成し、まず、このアルゴリズムに対して検証を行う。そして、アルゴリズムをＡＳＩＣ等のハードウェアにより実現する部分と、ソフトウェアにより実現する部分へ分割する。その後、ハードウェア部分についてはＲＴＬ　ＨＤＬ記述を作成し、ソフトウェア部分についてはＣＰＵモデルを作成する。さらに、トップレベルＨＤＬ記述を作成した上で、これらのＨＤＬ記述及びＣＰＵモデルに基づきＲＴＬ・ＨＤＬコシミュレーション装置によって、シミュレーションが行われる。

【０００３】

他方、アルゴリズム記述のシミュレーションよりも精細に、かつＲＴＬ　ＨＤＬ記述より高速にシミュレーションすることができるシミュレーションモデルを

構築するために、クロックベースシミュレーション技術が提案されている。例えば、特開２００１－１０９７８８号公報や「ＳＯＣの事前検証を実現するＣ＋＋シミュレータ」（黒川秀文著、信学技報VLH98-46）に、このクロックベースシミュレーション技術が開示されている。このクロックベースシミュレーションでは、クロックベース記述に基づき、シミュレーションが実行される。クロックベース記述は、アルゴリズムレベルより抽象度が低く、ＲＴＬ　ＨＤＬ記述よりも抽象度が高い。このクロックベース記述においては、バスが抽象化されており、バスへのアクセスが統一化されている。そして、バス・マスタのターゲットに対するアクセスは抽象的なバス・クラスを介して行われる。

【０００４】

【発明が解決しようとする課題】

従来のシステムレベル設計方法では、人手によって設計する部分が多く、労力を要し、かつ短期間で設計を行うという要請に応えることは困難であった。特に、クロックベースシミュレーション用記述の自動生成は実現されていない。

【０００５】

本発明は、このような問題を解決するためになされたものであり、クロックベースシミュレーション用の記述を自動的に生成することができるシステムレベル設計方法及び装置を提供することを目的とする。

さらに、アルゴリズムレベル記述をコデザイン用のアルゴリズムレベル記述に変更する際に、回路規模を縮小することが可能なシステムレベル設計方法及び方法を提供することを目的とする。

【０００６】

【課題を解決するための手段】

本発明にかかるシステムレベル設計方法は、コデザイン装置によってシステムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するステップと、前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報をトップレベル記述生成装置に入力するステップと、前記トップレベル記述生成装置において入力された情報

に基づいてクロックベースシミュレーション用の記述を自動生成するステップとを備えたものである。このような方法により、クロックベースシミュレーション用の記述を自動的に生成することができる。

【０００７】

ここで、クロックベースシミュレーション用の記述を自動生成するステップは、前記アドレス情報に基づき、バスとＣＰＵインターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成するステップを含むことが望ましい。

【０００８】

また、クロックベースシミュレーション用の記述を自動生成するステップにおいては、仮想バスによりバス接続の記述が行われていることが好ましい。

【０００９】

さらに、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するステップを備え、このステップは、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、検出された場合に当該グローバル変数をポートに置き換えるステップを有するとよい。このような方法により回路規模を縮小できる。

【００１０】

本発明にかかる他のシステムレベル設計方法は、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計方法であって、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出するステップと、検出された場合に当該グローバル変数をポートに置き換えるステップを備えたものである。このような方法により回路規模の縮小と、メモリを介する転送でないため高速処理が期待できる。

【００１１】

他方、本発明にかかるシステムレベル設計装置は、システムのハードウェア部分とソフトウェア部分を分割するとともに、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成するコデザ

イン装置と、前記データベースよりビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報を含む情報を入力し、入力した情報に基づいてクロックベースシミュレーション用の記述を自動生成するトップレベル記述生成装置を備えたものである。このような構成により、クロックベースシミュレーション用の記述を自動的に生成することができる。

【００１２】

ここで、前記トップレベル記述生成装置は、クロックベースシミュレーション用記述を自動生成する場合に、前記アドレス情報に基づき、バスとＣＰＵインターフェース間にあるアルゴリズムブロックの選択のためのアドレスデコーダに相当する記述を自動生成する。

【００１３】

また、トップレベル記述生成装置は、仮想バスによりバス接続の記述を行うとよい。

【００１４】

さらに、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換する手段を備え、この手段は、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出する手段と、検出された場合に当該グローバル変数をポートに置き換える手段とを有するようにするとよい。このような構成により回路規模の縮小と、メモリを介する転送でないため高速処理が期待できる。

【００１５】

本発明にかかる他のシステムレベル設計装置は、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述に自動変換するシステムレベル設計装置であって、アルゴリズムブロック間でグローバル変数を介して一方向にのみデータが流れる構成を検出する手段と、検出された場合に当該グローバル変数をポートに置き換える手段を備えたものである。このような構成により回路規模を縮小できる。

【００１６】

【発明の実施の形態】

本発明にかかるシステムレベル設計装置の構成を図１に示す。

図１に示されるように、当該システムレベル設計装置は、アルゴリズム記述１を検証するシステムレベルアルゴリズム検証装置２を備えている。このシステムレベルアルゴリズム検証装置２は、アルゴリズム記述１をシステムレベルアルゴリズム検証ツールにより検証する。一般にアルゴリズム記述は、プログラミング言語であるＣ言語やＣ＋＋言語によって表現される。システムレベルアルゴリズム検証ツールは、例えば、ケーデンス社のＳＰＷやシノプシス社のＣＯＳＳＡＰ等のツールである。

【００１７】

アルゴリズム検証装置２におけるアルゴリズム記述１の例を、図２（ａ）に示す。図２（ａ）において、Ｂ１乃至Ｂ７のそれぞれが、アルゴリズム記述をしたブロックである。また、ａ、ｂ、ｃ、ｄがグローバル変数を示している。ブロックＢ１の信号は、ブロックＢ２に入力されている。ブロックＢ２は、グローバル変数ａ及びｂを参照するとともに、グローバル変数ｂ、ｃ及びｄに対して代入している。また、ブロックＢ２は、ブロックＢ３及びＢ４へ信号を出力している。ブロックＢ３は、グローバル変数ｂに対して代入し、かつブロックＢ５に信号を出力している。ブロックＢ５は、グローバル変数ｃ、ｄを参照するとともに、グローバル変数ａ、ｂに対して代入している。また、ブロックＢ５は、信号をブロックＢ６に出力している。ブロックＢ４は、信号をブロックＢ７に出力している。このように、アルゴリズム検証装置２におけるアルゴリズム記述１では、一部の信号だけが各アルゴリズムのポートとして表現され、殆どの通信データは、グローバル変数で表現することができる。一部の信号は、大抵ブロックが終了して次のブロックを起動するためのトリガとなる。

【００１８】

本発明にかかるシステムレベル設計装置は、さらにアルゴリズム記述１を、ハードウェア（ＨＷ）とソフトウェア（ＳＷ）に分割しやすいようなコデザイン用アルゴリズム記述４に変換するコデザイン用アルゴリズム変換装置３を備えている。このコデザイン用アルゴリズム変換装置３は、上述のシステムレベルアルゴリズム検証装置２によってアルゴリズム上問題がないことが確認できた後に処理

を開始する。

【００１９】

コデザイン用アルゴリズム変換装置３により生成された、ＨＷとＳＷに分割しやすいようなコデザイン用アルゴリズム記述４は、コデザイン装置５に対して入力される。コデザイン装置５は、ＨＷ／ＳＷ分割装置７を有する。このＨＷ／ＳＷ分割装置７は、アルゴリズム記述をしたブロックのどれをＨＷとして表現し、どれをＳＷとして表現するか等を検討しながら、アーキテクチャを決定する。即ち、ＨＷ／ＳＷ分割装置７は、ＨＷとＳＷを分割する。ここで、ＨＷは例えばＡＳＩＣにより構成され、ＳＷはＣＰＵ上に構成される。コデザイン装置５は、さらに性能シミュレーション装置８を備えている。この性能シミュレーション装置８は、指定するアーキテクチャによって目的の性能が得られるかを検証するために、性能シミュレーションを実行する。

【００２０】

具体的には、性能シミュレーション装置８は、性能シミュレーションのために、まずアーキテクチャダイアグラムを用意する。アーキテクチャダイアグラムは、アーキテクチャインスタンスとして、ＣＰＵ、ＡＳＩＣ、ＲＴＯＳ、ＭＥＭＯＲＹ、ＢＵＳモデル等を使用してアーキテクチャの全体構成を表現したものである。性能シミュレーション装置８は、アーキテクチャインスタンス上で動作させるアルゴリズムを変えながら、遅延を考慮したシミュレーション、即ち性能シミュレーションを実行し、目的の性能としてふさわしいアーキテクチャか判断する。

【００２１】

コデザイン装置５で使用するアルゴリズムは、システムレベルアルゴリズム検証装置２において用いた記述から不必要なグローバル変数を除去して、関数の引数、即ちアルゴリズムのポートとしたものである。このような処理は、具体的には、コデザイン用アルゴリズム変換装置３において実行される。アルゴリズムがアーキテクチャの中で１つのブロックで表現されるとき、関数のような表現をとっており、グローバル変数を引数にしておけば、それをポートとして扱う。例えば、図２（ａ）で示されるような記述は、図２（ｂ）で示されるような記述に変

更する。このようにポートで表現することによりアルゴリズム間の記憶素子を削減でき、回路規模を小さくすることができる。本発明にかかるシステムレベル設計装置は、このようにグローバル変数を除去する処理に１つの特徴を有する。その処理の具体的な説明については、後に詳述する。

【００２２】

コデザイン装置５においてアーキテクチャの決定が終了すると、そのデータは、トップレベル記述生成装置９、ＨＷ部生成装置14及びＳＷ部生成装置２１に渡る。このデータには、ビヘイビア情報、アーキテクチャ情報、マッピング情報及びアドレス情報が含まれる。

【００２３】

トップレベル記述生成装置９は、ＨＷとＳＷの両方を含むアーキテクチャ全体を表現するトップレベル記述を生成する。トップレベル記述生成装置９は、トップＨＤＬ生成装置１０とクロックベース記述生成装置１２を備えている。

【００２４】

トップＨＤＬ生成装置１０は、コデザイン装置５によって決定されたアーキテクチャ情報等の情報に基づき、後述するＲＴＬ　ＨＤＬコシュミレーション装置２６へのトップレベル記述１１を生成する。

【００２５】

クロックベース記述生成装置１２は、コデザイン装置５によって決定されたアーキテクチャ情報等の情報に基づき、後述するクロックベースシミュレーション装置２５へのトップレベル記述１３を生成する。

【００２６】

ＨＷ部生成装置14は、インターフェース合成装置１５、動作合成装置１８を備えている。このインターフェース合成装置１５は、コデザイン装置５によって決定されたアーキテクチャ情報に基づき、アーキテクチャのＨＷ部記述生成を行い、ＨＷ部のアルゴリズム記述１６を生成する。さらに、インターフェース合成装置１５は、ＨＷ部とＢＵＳをつなぐインターフェース記述を合成し、インタフェース記述１７を生成する。動作合成装置１８は、ＨＷ部のアルゴリズム記述１６とインタフェース記述１７に基づいて、後述するＲＴＬ　ＨＤＬコシュミレー

ション装置へのＲＴＬ　ＨＤＬ記述１９を生成する。また、動作合成装置１８は、ＨＷ部のアルゴリズム記述１６とインターフェース記述１７に基づいて、後述するクロックベースシミュレーション装置２５へのクロックベース記述２０を生成する。

【００２７】

ＳＷ部生成装置２１は、ソフトウェア合成装置２２を備えている。このソフトウェア合成装置２２は、コデザイン装置５によって決定されたビヘイビア情報、アーキテクチャ情報、マッピング情報等の情報に基づき、ＲＴＬ　ＨＤＬコシミュレーション装置２６において動作できるＣＰＵモデル２３を生成する。また、ソフトウェア合成装置２２は、コデザイン装置５によって決定されたビヘイビア情報、アーキテクチャ情報、マッピング情報等の情報に基づき、クロックベースシミュレーション装置２５において動作できるＣＰＵモデル２４も生成する。

【００２８】

トップレベル記述生成装置９によって生成されたＨＤＬ記述１１、ＨＷ部生成装置１４によって生成されたＲＴＬ　ＨＤＬ１９及びＳＷ部生成装置２１によって生成されたＣＰＵモデル２３は、ＲＴＬ　ＨＤＬコシミュレーション装置２６に入力される。ＲＴＬ　ＨＤＬコシミュレーション装置２６は、ＲＴＬ　ＨＤＬ記述レベルにおいて、全体的なシミュレーションを実行する。

【００２９】

トップレベル記述生成装置９によって生成されたクロックベース記述１３、ＨＷ部生成装置１４によって生成されたクロックベース記述２０及びＳＷ部生成装置２１によって生成されたＣＰＵモデル２４は、クロックベースシミュレーション装置２５に入力される。クロックベースシミュレーション装置２５は、クロックベース記述レベルにおいて、全体的なシミュレーションを実行する。ここで、クロックベース記述は、ＲＴＬ　ＨＤＬ記述より上位レベルであり、アルゴリズム記述より下位レベルの記述である。そのため、クロックベースシミュレーション装置２５は、ＲＴＬ　ＨＤＬコシミュレーション装置２６によって実行されるＨＤＬコシミュレーションよりも高速なシミュレーションを実行することができる。そのアルゴリズムの内容によるが、クロックベース記述のシミュレーション

時間は、ＲＴＬ　ＨＤＬ記述のシミュレーション時間の概ね５００分の１である。

【００３０】

クロックベースシミュレーションでは、例えば、次のような項目の見積もり及び検証を行うことができる。

（１）各モジュールのクロックベースの動作タイミング検証

（２）各モジュールのインターフェースの概略検証（ＩＯレジスタ、ＩＯメモリ、ＩＯ端子の構成、名称、ビット幅等）

（３）各モジュール、バスの動作クロックの周波数見積もり

（４）キャッシュアクセスの見積もり（キャッシュヒット率、アクセス率、ライトバック回数等）

（５）アクセスの見積もり（バスの占有率、バスのトランザクションごとのＡｄｒｓ、Ｄａｔａ、Ｍａｓｔｅｒ、Ｓｌａｖｅ、Ｒｅａｄ／Ｗｒｉｔｅ、Ｃｏｍｍａｎｄ、語数、占有時間等）

（６）Ｂｕｓ、Ａｒｂｉｔｅｒのアルゴリズムの検証

（７）メモリ／ＩＦのトラフィックの見積もり

（８）データ処理のスループットの見積もり（モジュール毎、バス毎、システム全体）

（９）バッファ、スタックサイズの見積もり

（１０）画質、音質の見積もり

（１１）アドレスマップ、端子、ｂｉｔ幅等モジュール間接続Ｉ／Ｆの整合性の検証

（１２）浮動小数点を固定小数点に変換した場合の見積もり

（１３）組み込みソフトウエアの開発、デバッグ

（１４）消費電力の概略見積もり

【００３１】

また、クロックベース記述のシミュレーションは、ＲＴＬ　ＨＤＬ記述のシミュレーションに比べて、幾つかのシミュレーションが省略されること等によってそのシミュレーション時間を高速化することができる。例えば、ＲＴＬ　ＨＤＬ

記述は、クロック信号の変化に応じてレジスタ値は新しいデータ入力値又はそのレジスタが保持していた値に常に更新されるようになるが、クロックベース記述では新しいデータ入力があった変数のみが更新される。また、ＲＴＬ　ＨＤＬモデルは、演算器の共有が行われるため、入力となる複数のレジスタから１つのレジスタを選択するマルチプレクサが必要であるが、クロックベース記述では演算器に制限がなく、演算器の入力を選択するマルチプレクサ及び制御入力を作成する回路が不要である。さらに、また、ＲＴＬ　ＨＤＬ記述は、全レジスタに非同期リセット信号が必要だが、クロックベース記述ではクロック周期単位の動作のみを扱うので、非同期的な動作を行う必要がない。また、ＲＴＬ　ＨＤＬ記述は、レジスタ、マルチプレクサ、演算器等のサブモジュールを利用した構造を持ち、そのサブモジュールは端子を持ち、その内部には信号線、制御機構等も持っているが、クロックベース記述では、レジスタはプログラミング言語の変数に、マルチプレクサは条件文に、演算器は演算子で表現される。また、ＲＴＬ　ＨＤＬ記述は、実際のハードウエア通りに束線のビット位置の昇降順、符号のありなし、整数型／ビットベクタ型等の信号型の区別、又は、これらの間の変換を厳密に表現する必要があるが、このような区別は、クロックベース記述では厳密に行われない。さらに、ＲＴＬ　ＨＤＬ記述内にある各サブモジュール間の動作の並列性は厳密に表現されるが、クロックベース記述では各サブモジュールの厳密な動作タイミングの差に基づいた検証は行われないため、並列性を厳密に表現する必要がない。また、ＲＴＬ　ＨＤＬ記述は、クロックの変化タイミング以外にも例えばリセット信号が変化した時点の動作を厳密に表現するが、クロックベース記述ではクロック単位の動作のみを扱うので、非同期的な動作に対する処理を簡略化することができる。

【００３２】

以上説明したようなシステムレベル設計装置により、システムレベルアルゴリズム記述から、アーキテクチャを決定してクロックベースレベルでの検証まで行う設計フローを実現する。

【００３３】

続いて、本発明にかかるシステムレベル設計装置の特徴的な構成及び処理につ

いて詳述する。

【００３４】

最初に、システムレベルアルゴリズム記述からコデザイン用システムレベルアルゴリズム記述への変換処理について説明する。この変換処理は、コデザイン用アルゴリズム変換装置３によって実行される。例えば、図２（ａ）に示すアルゴリズムは、図２（ｂ）に示すアルゴリズムのようにできるだけグローバル変数を減らすように変換される。

【００３５】

この変換処理においては、第１に、図２（ａ）に示すａ、ｃ、ｄのように一方向にしたデータが流れない場合には、単純にポートに変換する。ポートに変換した後の構成を図２（ｂ）に示す。図２（ｂ）に示されるように、アルゴリズムブロックＢ２とアルゴリズムブロックＢ５間にポートを設定している。ただし、図２（ａ）に示すｂについては、複数のアルゴリズムＢ２、Ｂ３、Ｂ５よりアクセスされているため、ポートに変換することはできない。

【００３６】

第２に、解消できないグローバル変数は、Behavioral memory（ビヘイビアレベルのメモリ）に変換する。図２（ａ）に示すｂのようにいたるところからアクセスされる場合、グローバル変数をメモリのような記憶素子で表現する。このような場合、回路はメモリで表現されるのが一般的である。複数のモジュールから一つの変数を書き込んでいき、あるタイミングで複数のモジュールから変数値の読み込みができる。

【００３７】

グローバル変数を減らす処理を行わずにコデザイン装置５のＨＷ／ＳＷ分割装置７によってＨＷ部とＳＷ部の分割処理を行った後のアーキテクチャを図３に示す。この構成は、図２（ａ）に相当するものであり、グローバル変数ａ、ｂ、ｃ、ｄがそれぞれメモリ３９、３３、４０、４１として表されている。図３においては、ＨＷ部３０とＳＷ部３１にアルゴリズムがマッピングされている。ＨＷ部３０では、書き込み手段３４及び読み込み手段３５を介して、バス３２とＢ２インターフェース３６、Ｂ４インタフェース３７及びＢ５インタフェース３８が接

続されている。Ｂ２インターフェース３６、Ｂ４インタフェース３７及びＢ５インタフェース３８は、インターフェース合成により自動生成され、それぞれ、アルゴリズムブロックＢ２、Ｂ４、Ｂ５と接続されている。ＳＷ部３１には、アルゴリズムブロックＢ１、Ｂ３、Ｂ６、Ｂ７がマッピングされている。また、バス３２には、直接、グローバル変数ｂに相当する外部メモリが接続されている。

【００３８】

図４にグローバル変数を減らす処理を行った場合の構成を示す。図４に示す通り、グローバル変数ａ、ｃ、ｄに相当するメモリは削除され、ポートに変換されている。そのため、図３に示すグローバル変数を減らす処理を行う前に比べて、メモリａ、ｃ、ｄ分だけ回路規模を小さくすることができる。

【００３９】

次に、グローバル変数削除処理のアルゴリズムの一例を説明する。図６に、記述例を示す。図６（ａ）は、図２（ａ）に示すアルゴリズムブロックＢ２、Ｂ３、Ｂ５を表現する記述である。まず、図６（ａ）の記述を解析し、グローバル変数のreadを使用しているか、writeを使用しているかを調査する。調査結果が図６（ｂ）である。次に、図６（ｂ）におけるread、writeの数の総計を計算する。グローバル変数のread、writeの数がそれぞれ１以下の場合は、ポートに変換し、グローバル変数は削除できる。この例では、ａ、ｃ、ｄがその変数に相当し、削除することができる。削除できずに残ったグローバル変数ｂは、そのままの状態が維持される。最終的に、関数は、図６（ｃ）に示す記述のように変換される。

【００４０】

続いて、コデザイン装置５からクロックベースシミュレーション装置２５へ接続するためのクロックベース記述生成装置１２の処理について詳述する。

【００４１】

まず、コデザイン装置５では、性能シミュレーションによる性能見積りを行いながらアーキテクチャ決定を行う。ここでは、Cadence社製のＶＣＣ（Virtual Component Codesign）の機能を前提に記述する。次にこのアルゴリズムを動作させるアーキテクチャモデル（アーキテクチャダイアグラム）を用意する。アーキ

テクチャはコデザイン装置上でキャラクタライズされた、プロセッサやバス等の様々なアーキテクチャコンポーネントを配置、結線して表現する。

【００４２】

次にマッピングを行う。アルゴリズムの各ブロックをアーキテクチャのどのコンポーネントで実行するか、アルゴリズムのデータの流れがアーキテクチャのどのインターフェースを流れるのかをＧＵＩ等を通して、線で結んでマッピングする。

【００４３】

通信経路の決定にはパタンを使用する方法もある。パタンは、アルゴリズムブロック間のデータ転送に関するプロトコルを表す。ＨＷ→ＳＷ間の通信であれば、ＣＰＵ　Ｉ／Ｆに相当するメモリマップＩＯのアドレス情報なども有する。

【００４４】

マッピングした後、個々のアーキテクチャコンポーネントに定義された遅延値がアルゴリズムにアノテートされ、シミュレーションを行うことにより、そのアルゴリズムがそのアーキテクチャにマッピングされたときの性能を検証できる。

【００４５】

クロックベース記述生成装置１２は、このようにして生成されたコデザイン装置５の情報を用いて、トップレベルクロックベース記述１３を作成する。

【００４６】

次に、クロックベース記述生成装置１２によりトップレベルクロックベース記述１３の作成につき説明する。トップレベルクロックベース記述１３は、次のようなフォーマットを有する。

インスタンス宣言；

Initialization処理；

Reset処理；

Step処理；

busread処理；

buswrite処理；

ブロック間の直接接続宣言；

【００４７】

このようなフォーマットの記述をクロックベース生成装置１２によって自動生成している。以下にそれぞれのフォーマットの生成につき説明する。

【００４８】

最初にインスタンス宣言の生成について説明する。トップレベルクロック記述におけるインスタンス宣言は、コデザイン装置５により生成されたマッピング情報から、アーキテクチャインスタンス、アルゴリズムインスタンス（ビヘイビアインスタンス）を取得することにより生成する。ここで出力するインスタンス宣言の対象は、プロセッサ、バスなどの既存アーキテクチャインスタンス、ＡＳＩＣにマッピングされたアルゴリズム、ＡＳＩＣにマッピングされたアルゴリズムのＣＰＵインターフェースである。コデザイン装置５におけるマッピングが図５に示すような構成を有する場合において、クロックベース生成装置１２は、図４に示すような構成のトップレベル記述を生成する。

【００４９】

ここで、アーキテクチャダイアグラム中に存在するインスタンスから作成されるインスタンス宣言例を以下に示す。

【００５０】

バスの場合には、次のようなインスタンス宣言となる。

```
BUS Bus_1("Bus1", arbitor_1, busread_1, buswrite_1);
int arbitor_1(void);
RDATA busread_1(int add, int byte_count);
int buswrite_1(int add, int data, int byte_count);
```

バスのインスタンス自体はBUS bus_1のように宣言し、arbitorの宣言も同時に出力する。arbitor_1の内容については適当なものを自動生成し、ユーザが変更したい場合、あとで本トップレベル記述を編集する。同時にbusread_1、buswrite_1といった関数宣言をする。これは図８、図９に示す記述のうち、busread_1、buswrite_1と記述したブロックに相当し、バスとＡＳＩＣ中のアルゴリズムのＣＰＵ　Ｉ／Ｆを結ぶ関数である。具体的には、メモリマップ方式におけるビヘイビア選択のためのアドレスがここに設定される。

【００５１】

プロセッサの場合には、次のようなインスタンス宣言となる。

```
CPU cpu_1("cpu_1",&cpu_1_busif,0,0x10000,0x00ffe000,0x1000,0x8000);
BusIntf cpu_1_busif("cpu 1 Bus I/F",&bus_1,1)
```

このようにしてＣＰＵコアの設定及びＣＰＵ用バスの設定を行う。数字は、ＲＯＭ、ＲＡＭのスタートアドレスに相当するが、本データに関してはコデザイン装置で扱っていない場合、未設定で出力し、ユーザが後に設定する。

【００５２】

外部ＲＯＭ、ＲＡＭの場合には、次のようなインスタンス宣言となる。

```
SDRAM ExtSDRam1("External SDRam 1" , EXTSDRAM1_START_ADD ,EXTSDRAM1_SIZE
)
```

コデザイン装置５におけるメモリのアーキテクチャインスタンスがこの宣言に変換される。

【００５３】

ＡＳＩＣにマッピングされたアルゴリズム、ＣＰＵ　Ｉ／Ｆは以下のような宣言で作成する。

```
c_cpuifB2 cpuifB2(); c_cpuifB4 cpuifB4(); c_cpuifB5 cpuifB5();
c_B2 funcB2(); c_B4 funcB4(); c_B5 funcB5();
```

ここで、c_cpuif***（***には任意の関数名が入る）はそれぞれのcpuifを表現するclassである。また、c_B**（**には任意のアルゴリズムの番号が入る）はそれぞれのalgorithmを表現するclassである。これらのclassや実体はＨＷ部生成装置１４で作成されるクロックベース記述である。

【００５４】

次にInitialization処理、Reset処理及びStep処理の生成について説明する。

アルゴリズム記述は動作合成によってクロックベース記述に合成される。クロックベース記述は、init()、reset()、OneStep()等の構成要素をもつ。init()は、対応するアルゴリズムブロックを最初に初期化するときに使用する。reset()は、対応するアルゴリズムブロックがリセットされた場合の内部変数の初期設定を行う。OneStep()は、アルゴリズムブロックに対してクロックが入った場合の

１クロック分の処理を記述する。このOneStep()は、アルゴリズム記述の中核となる関数である。以上のようなブロックを並列に動作させる場合、以下のように使用する。

```
{
funcB2.OneStep(); cpuifB2.OneStep();
funcB4.OneStep(); cpuifB4.OneStep();
funcB5.OneStep(); cpuifB5.OneStep();
}
```

この記述は、Ｂ２、Ｂ４、Ｂ５のアルゴリズムブロックが並列動作することを意味する。初期化、リセットも同様に次のような記述がされる。

```
systemInit()
{
funcB2.Init(); cpuifB2. Int();
funcB4.Init(); cpuifB4. Int();
funcB5.Init(); cpuifB5. Int();
}
systemReset()
{
funcB2.Reset(); cpuifB2. Reset();
funcB4.Reset(); cpuifB4. Reset();
funcB5.Reset(); cpuifB5. Reset();
}
```

【００５５】

次にbusread処理及びbuswrite処理の生成について説明する。busread及びbuswriteは、コデザイン装置５によって生成されるマッピング情報よりアドレス情報を取得することによって作成される。busread及びbuswriteは、バスとＣＰＵＩ／Ｆの間にあるアルゴリズムデコーダに相当する。cpuifとbusread、buswriteとの関係は図８及び図９のように表すことができる。図中にあるＣＰＵ　Ｉ／Ｆはインターフェース合成及び動作合成により作成される。インターフェース合成

により入力記述は、特開２００１－１１７８５５号公報に示される手法により使用するものを前提としている。また、アルゴリズム部はアルゴリズム記述から動作合成によって作成されるクロックベース記述である。このように、本発明にかかるシステムレベル設計方法では、コデザイン装置５によって生成されるマッピング情報よりアドレス情報を取得することによってbusread及びbuswriteを作成している点に特徴を有する。ＨＤＬ記述においては、バス構造、バスの使用方法に応じたピンを指定しなければならかったため、この処理を自動化することは困難であった。これに対して、クロックベース記述は、仮想バスに指定すればよく、ピン情報を詳細に設定する必要がないため、アドレス情報に基づきbusread及びbuswriteを自動的に作成することができる。また、通常は、バスとＣＰＵ　Ｉ／Ｆの間を直接的に関連付けているが、本発明にかかるシステムレベル設計方法では、busread及びbuswriteというアドレスデコーダに相当する記述を設けており、この点にも特徴を有する。

【００５６】

続いて、ブロック間の直接接続宣言の生成について説明する。

図１０に示すアルゴリズムブロックＢ２とＢ５の間の直接接続記述は以下のように作成される。

```
connection()
{
B5.a = B2.a;
B5.b = B2.b;
B5.c = B2.c;
}
```

ここでは説明を省くが、ＣＰＵ　Ｉ／Ｆとアルゴリズムブロック間の直結にも直接接続は使用される。

【００５７】

ここで、トップレベルＨＤＬ記述とトップレベルクロックベース記述の違いについて詳述する。

基本的に両者が大きく異なる点は、バス接続の記述方法である。クロックベー

ス記述の場合、cpuifのReadIOReg(), WriteIOReg()でＣＰＵ　Ｉ／Ｆとのデータのやり取りを行い、RDATAという構造体でbusにデータを渡す。

ＨＤＬ記述の場合、図１１に示すようにバスと物理的なピンで接続するモデルとなっている。バスが変更されるとピン情報が変わり、接続の仕方もその都度変えなければならない。また、詳細なプロトコルを表現することも可能である。

クロックベース記述の場合、RDATAという構造体でデータを渡すため、図１０に示すように、バスの物理的な構造によらずにデータのやり取りを表現する。細かいバスプロトコルを考慮できないため、バスの使用の仕方によっては、ＲＴＬＨＤＬ程の正確な遅延見積りがこの部分で行なえないかもしれない。ユーザはその点を考慮にいれて使用する必要がある。

【００５８】

このように本発明にかかるシステムレベル設計方法では、アルゴリズム記述中における冗長なグローバル変数表現をポート表現に変換することで、回路規模の縮小を図ることができる。また、コデザイン装置における記述からクロックベースシミュレーション装置用のクロックベース記述に自動変換している。特に本発明では、従来自動合成していなかったトップレベルクロックベース記述を自動合成することで、クロックベースシミュレーションを行なうための環境構築期間の短縮化を図ることができる。

【００５９】

【発明の効果】

本発明によれば、クロックベースシミュレーション用の記述を自動的に生成することができるシステムレベル設計方法及び装置を提供することができる。

また、アルゴリズムレベル記述をコデザイン用のアルゴリズムレベル記述に変更する際に、回路規模を縮小することができる。その理由は、アルゴリズム記述中における冗長なグローバル変数表現をポート表現に変換しているからである。

【図面の簡単な説明】

【図１】

本発明にかかるシステムレベル設計方法のフローを示す図である。

【図２】

システムレベルアルゴリズムからコデザイン用システムレベルアルゴリズムへの変換を説明するための図である。

【図３】

グローバル変数の変換を説明するための図である。

【図４】

グローバル変数の変換とトップレベルクロックベース記述を説明するための図である。

【図５】

コデザイン装置中のデータを示す図である。

【図６】

グローバル変数の変換アルゴリズムを示す図である。

【図７】

トップレベルクロックベース記述の一例を示す図である。

【図８】

トップレベルクロックベース記述の一例を示す図である。

【図９】

トップレベルクロックベース記述の一例を示す図である。

【図１０】

トップレベルクロックベース記述を説明するための図である。

【図１１】

トップレベルクロックベース記述を説明するための図である。

【符号の説明】

３　コデザイン用アルゴリズム変換装置

５　コデザイン装置

９　トップレベル記述生成装置

１２　クロックベース記述生成装置

１４　ＨＷ部生成装置

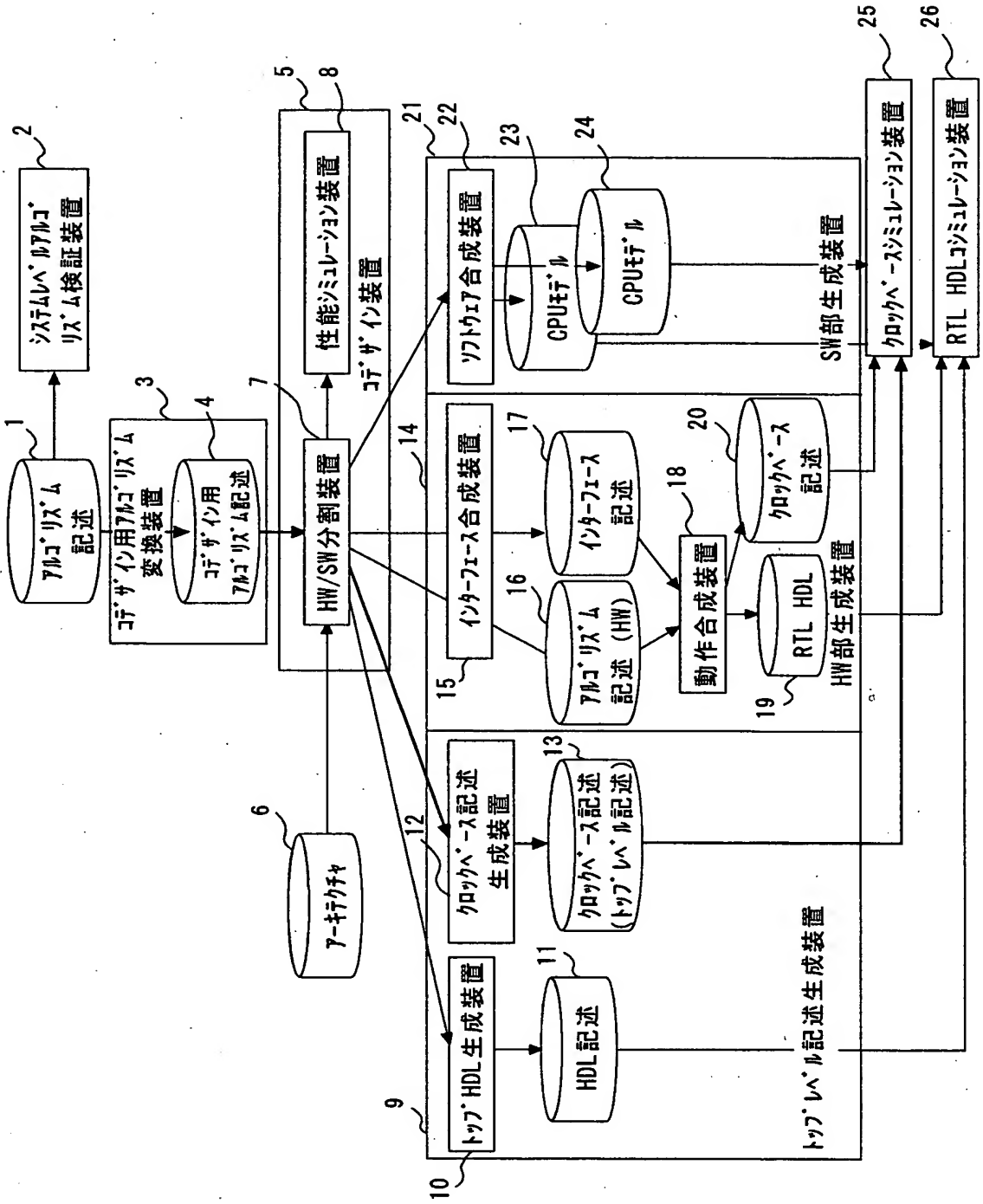
２１　ＳＷ部生成装置

２５　クロックベースシミュレーション装置

２６．ＲＴＬ　ＨＤＬコシミュレーション装置

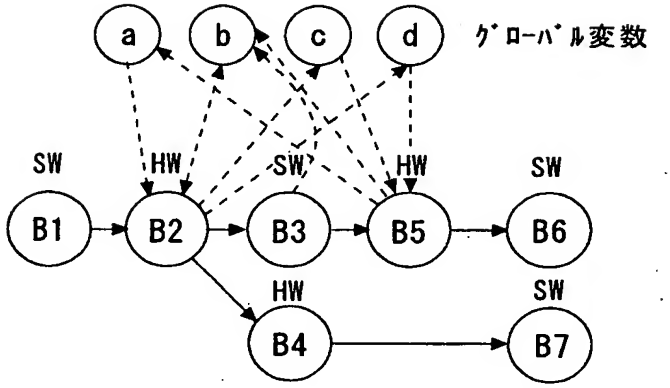
【書類名】　　図面

【図１】


1
アルゴリズム記述
2
システムレベルアルゴリズム検証装置
3
コデザイン用アルゴリズム変換装置
4
コデザイン用アルゴリズム記述
5
コデザイン装置
6
アーキテクチャ
7
HW/SW分割装置
8
性能シミュレーション装置
9
トップHDL生成装置
10
トップレベル記述生成装置
11
HDL記述
12
クロックベース記述生成装置
13
クロックベース記述（トップレベル記述）
14
インターフェース合成装置
15
アルゴリズム記述（HW）
16
インターフェース記述
17
動作合成装置
18
RTL HDL
19
HW部生成装置
20
クロックベース記述
21
ソフトウェア合成装置
22
CPUモデル
23
CPUモデル
24
SW部生成装置
25
クロックベースシミュレーション装置
26
RTL HDLシミュレーション装置

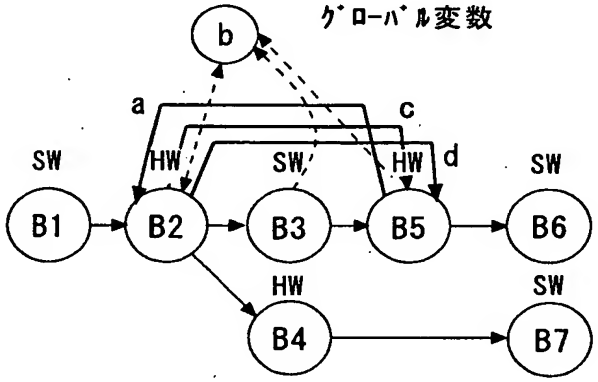
【図２】

（ａ）

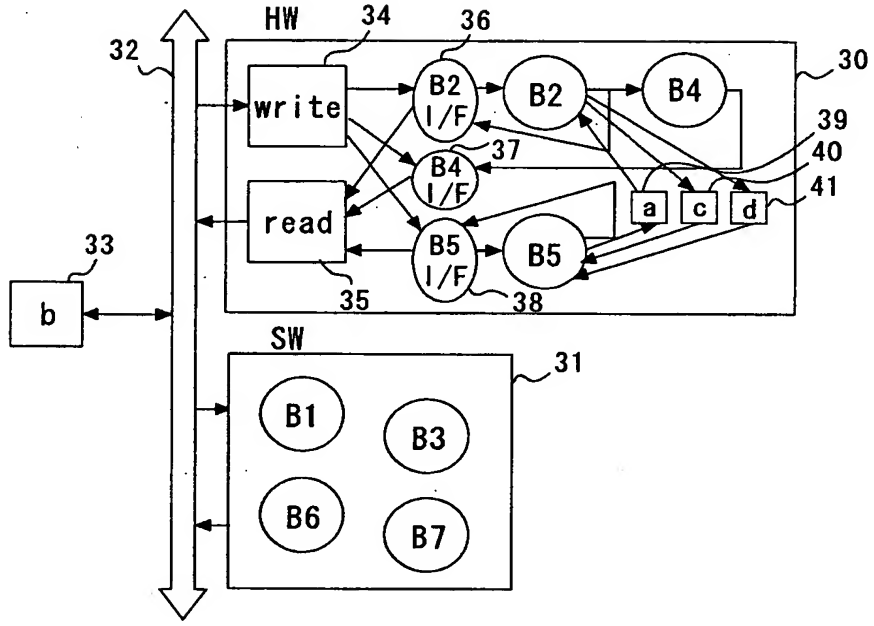


ｸﾞﾛｰﾊﾞﾙ変数の解消

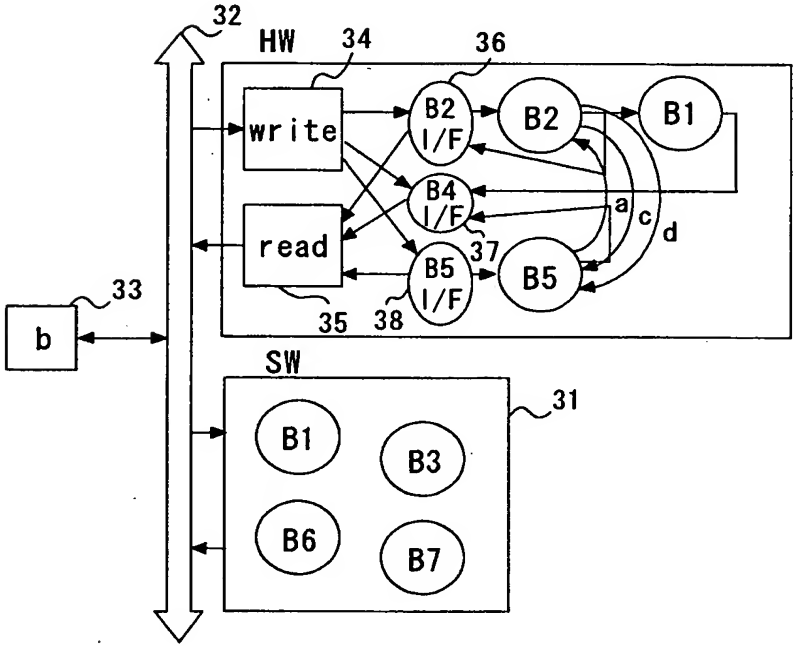
（ｂ）



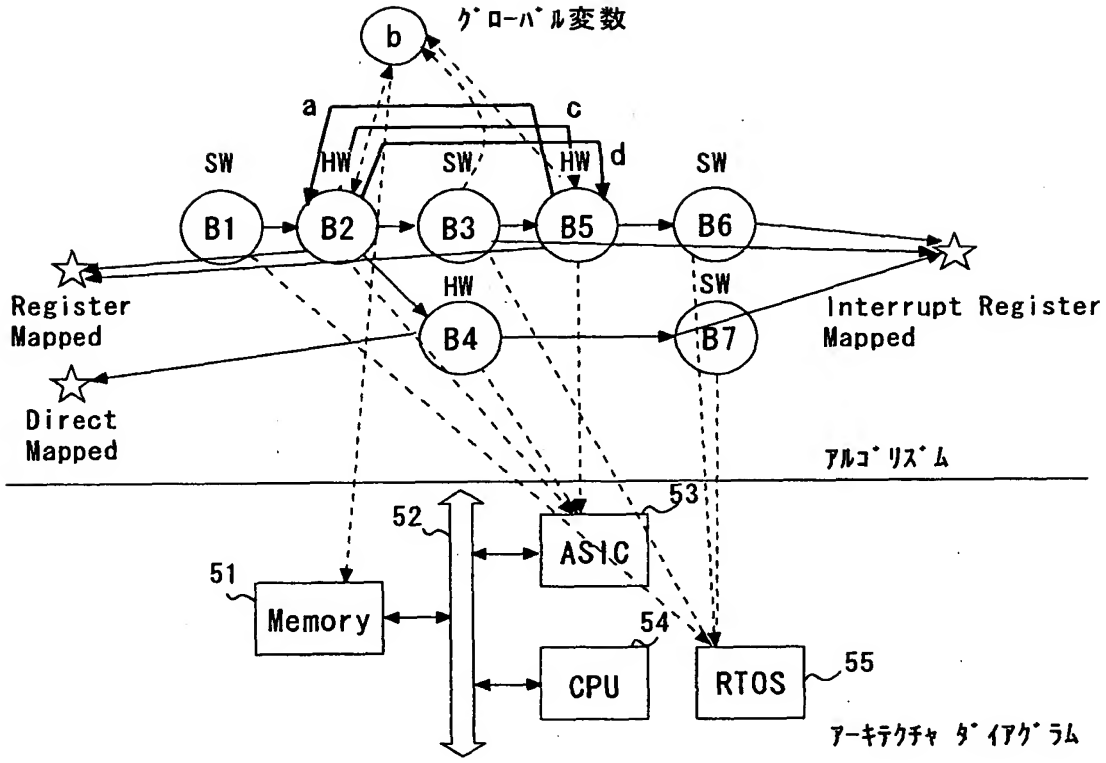
【図３】


HW
34
36
32
write
B2 I/F
B2
B4
30
39
40
41
B4 I/F
37
read
a
c
d
B5 I/F
B5
33
b
35
38
SW
31
B1
B3
B6
B7

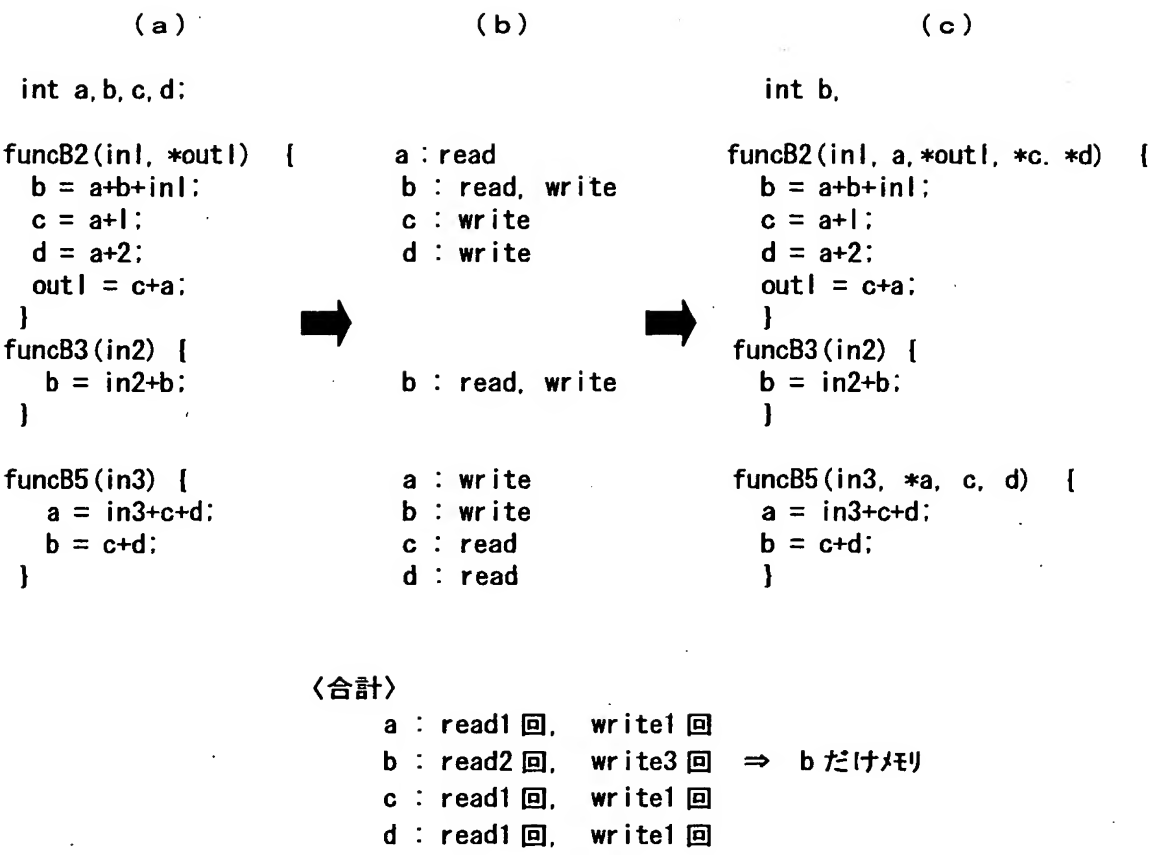
【図４】


32
HW
34
36
write
B2 I/F
B2
B1
B4 I/F
a
c
d
read
37
B5 I/F
B5
33
b
35
38
SW
31
B1
B3
B6
B7

【図５】


b
ｸﾞﾛｰﾊﾞﾙ変数
a
c
d
SW
HW
SW
HW
SW
B1
B2
B3
B5
B6
Register
Mapped
HW
SW
B4
B7
Interrupt Register
Mapped
Direct
Mapped
ｱﾙｺﾞﾘｽﾞﾑ
52
53
ASIC
51
Memory
54
CPU
RTOS
55
ｱｰｷﾃｸﾁｬ ﾀﾞｲｱｸﾞﾗﾑ

【図６】



〈合計〉

a : read1回, write1回
b : read2回, write3回 ⇒ bだけメモリ
c : read1回, write1回
d : read1回, write1回

【図７】

```
BUS_bus 1( "Bus1" , arbitor_1, busread_1, buswrite_1)
Int arbitor_1(void);
RDATA busread_1(int add, int byte_count);
Int buswrite_1(int add, int data, int byte_count);

CPU ( "cpu_1" , &cpu_1_busif, 0, 0X10000, 0X00ffe000, 0X00000X6000);
BusIntf cpu_1_busif( "cpu 1 Bus I/F , &bus_1, 1);

SDRAM ExtSDRam1( "External SDRam 1" , EXTSDRAM1_START_ADD, EXTSDRAM1_SIZE_);

c_cpuifB2 cpuifB2(); c_cpuifB4 cpuifB4(); c_cpuifB5 cpuifB5();
c_B2 funcB2(); c_B4 funcB4(); c_B5 funcB5();

SystemoneScep()
{
   funcB2.OneStep(); cpuifB2.OneStep() ;
   funcB4.OneStep(); cpuifB4.OneStep() ;
   funcB5.OneStep(); cpuifB5.OneStep() ;
}
SystemInit()
{
   funcB2.Init(); cpuifB2. Init();
   funcB4.Init(); cpuifB4. Init();
   funcB5.Init(); cpuifB5. Init();
}
SystemReset()
{
   funcB2.Reset(); cpuifB2. Reset();
   funcB4.Reset(); cpuifB4. Reset();
   funcB5.Reset(); cpuifB5. Reset();
}
```

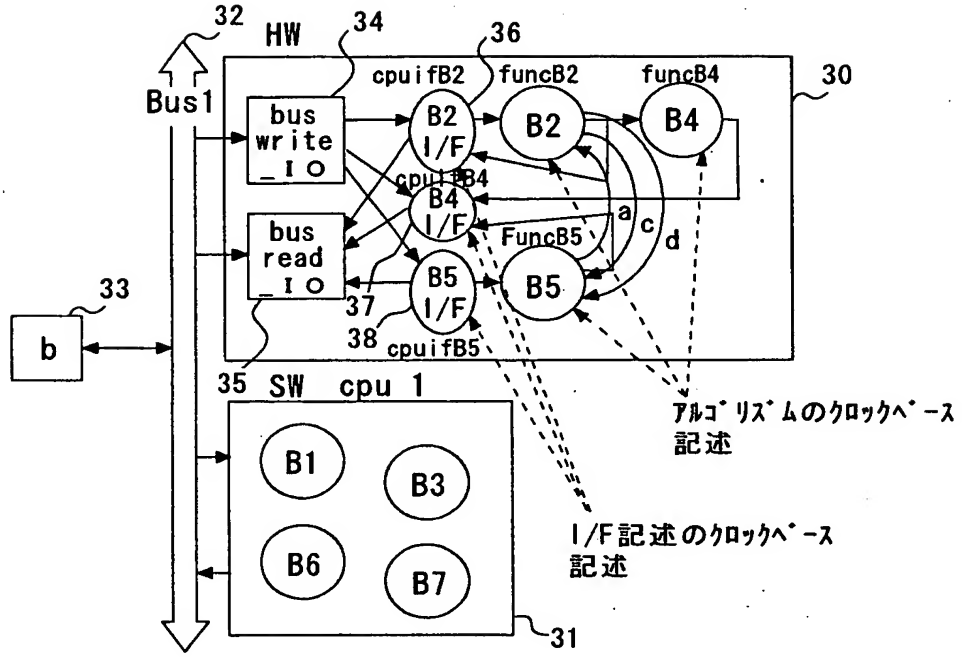
【図８】

```
RDATA busread_1 (int add, int byte_count)  {
  if ((add < start_bus_address) || (add > end_bus_address))  {
    return(NULL);
  }
  //RAM
  if ((add >= ExternalRAM.StartAddress) && (add < ExternalRAM. EndAddress))  {
    return ExternalRAM.Read(add - ExternalRAM.StartAddress, byte_count);
  }

  //ASIC
  if ((start_sub__address+B2_start_offset <= add)
       && (add < start_sub_address+B2_end_offset))  {
    cpuifB2. cpuread = 0;
    cpuifB2. bus_addin = add;
    rd.Status = cpuifB2.ReadIOReg();
    rd.data = cpuifB2.dataout;
   }
  if ((start_sub__address+B4_start_offset <= add)
       && (add < start_sub_address+B4_end_offset))  {
    cpuifB4. cpuread = 0;
    cpuifB4. bus_addin = add;
    rd.Status = cpuifB4 ReadIOReg();
    rd.data = cpuifB4.dataout;
  }
  if ((start_sub__address+B5_start_offset <= add)
       && (add < start_sub_address+B5_end_offset))  {
    cpuifB5. cpuread = 0;
    cpuifB5. bus_addin = add;
    rd.Status = cpuifB5.ReadIOReg();
    rd.data = cpuifB5.dataout;
  }

  ---------

  return (rd);
}
```

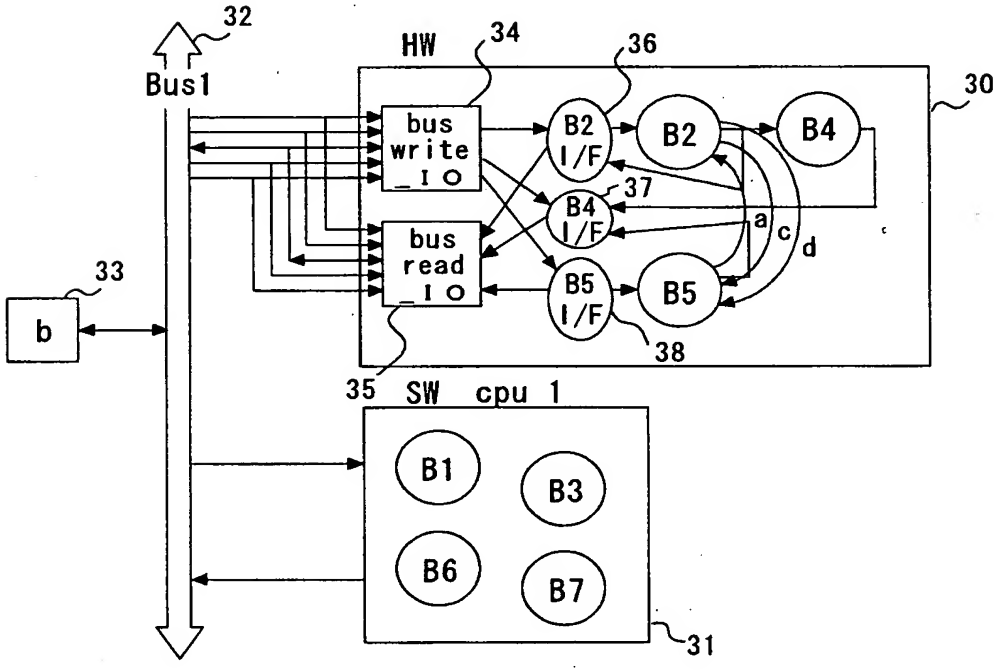
【図９】

```
RDATA buswrite_1 (int add, int data, int byte_count)  {
  if ((add < start_bus_address) || (add > end_bus_address))  {
    return(1);
  }
  //RAM
  if ((add >= ExternalRAM.StartAddress) && (add < ExternalRAM. EndAddress))  {
    return ExternalRAM.Write(add - ExternalRAM.StartAddress, byte_count);
  }

  //ASIC
  if ((start_sub__address+B2_start_offset <= add)
      && (add < start_sub_address+B2_end_offset))  {
    cpuifB2. cpuwrite = 1;
    cpuifB2. bus_addin = add;
    cpuifB2. datain = data;
    rd.Status = cpuifB2.WriteIOReg();
  }
  if ((start_sub__address+B4_start_offset <= add)
      && (add < start_sub_address+B4_end_offset))  {
    cpuifB4. cpuwrite = 1;
    cpuifB4. bus_addin = add;
    cpuifB4. datain = data;
    rd.Status = cpuifB4.writeIOReg();
  }
  if ((start_sub__address+B5_start_offset <= add)
      && (add < start_sub_address+B5_end_offset))  {
    cpuifB5. cpuwrite = 1;
    cpuifB5. bus_addin = add;
    cpuifB5. datain = data;
    rd.Status = cpuifB5.writeIOReg();
  }

  ----------

  return (0);
}
```

【図１０】


32
HW
34
36
Bus1
cpuifB2
funcB2
funcB4
30
bus write _IO
B2 I/F
B2
B4
cpuifB4
B4 I/F
a
c
d
bus read _IO
FuncB5
B5 I/F
B5
37
33
b
38
cpuifB5
35 SW cpu 1
B1
B3
B6
B7
31
アルゴリズムのクロックベース記述
I/F記述のクロックベース記述

【図１１】


32
Bus1
HW
34
36
30
bus write _IO
B2 I/F
B2
B4
37
B4 I/F
a
c
d
bus read _IO
B5 I/F
B5
38
33
b
35 SW cpu 1
B1
B3
B6
B7
31

【書類名】　　要約書

【要約】

【課題】

クロックベースシミュレーション用の記述を自動的に生成すること。

【解決手段】

本発明にかかるシステムレベル設計方法は、コデザイン装置５によってシステムのハードウェア部分とソフトウェア部分を分割するとともに、アーキテクチャ情報、マッピング情報及びアドレス情報を含むデータベースを自動生成する。その後、トップレベル記述生成装置９にこのデータベースよりアーキテクチャ情報、マッピング情報及びアドレス情報を含む情報を入力する。さらに、トップレベル記述生成装置９は、入力された情報に基づいてクロックベースシミュレーション用の記述を自動生成する。

【選択図】　図１

認定・付加情報

特許出願の番号	特願２００２－２１３８９５
受付番号	５０２０１０８１１７４
書類名	特許願
担当官	第七担当上席　００９６
作成日	平成１４年　７月２４日

＜認定情報・付加情報＞

【提出日】　平成14年 7月23日

次頁無

【書類名】　出願人名義変更届（一般承継）
【提出日】　平成15年 1月14日
【あて先】　特許庁長官殿
【事件の表示】
　【出願番号】　特願2002-213895
【承継人】
　【識別番号】　302062931
　【氏名又は名称】　ＮＥＣエレクトロニクス株式会社
【承継人代理人】
　【識別番号】　100103894
　【弁理士】
　【氏名又は名称】　家入　健
【提出物件の目録】
　【物件名】　承継人であることを証明する登記簿謄本　1
　　【援用の表示】　特願２００２－３１８４８８
　【物件名】　承継人であることを証明する承継証明書　1
　　【援用の表示】　特願２００２－３１８４８８
　【包括委任状番号】　0218232
【プルーフの要否】　要

出　願　人　履　歴　情　報

識別番号　　〔０００００４２３７〕

１．変更年月日　　１９９０年　８月２９日
［変更理由］　新規登録
住　所　東京都港区芝五丁目７番１号
氏　名　日本電気株式会社